(12)

NAVSWC TR 91-592

SYSTEMS MODELING

BY DONG CHOI JANET YOUNGBLOOD STEVE HOWELL PHIL HWANG

UNDERWATER SYSTEMS DEPARTMENT

2 FEBRUARY 1992

NAVAL SURFACE WARFARE CENTER

Dahlgren, Virginia 22448-5000 • Silver Spring, Maryland 20903-5000

NAVSWC TR 91-592

SYSTEMS MODELING

BY DONG CHOI JANET YOUNGBLOOD STEVE HOWELL PHIL HWANG
UNDERWATER SYSTEMS DEPARTMENT

2 FEBRUARY 1992

NAVAL SURFACE WARFARE CENTER
Dahlgren, Virginia 22448-5000 ● Silver Spring, Maryland 20903-5000

FOREWORD

Systems Modeling explores three modeling and analysis approaches that aid the development of naval systems: dynamic performance modeling, rapid prototyping, and executable specification. The use of these techniques in the development of large, complex systems helps designers predict the behavior of the system, assess the feasibility of the design, and seek the optimal solution from the design space. Further research is being directed toward improving the integration and advancement of current modeling techniques to provide the systems designer with an integrated, seamless transition between phases and analyses in the system development process.

The work to date was performed in conjunction with the Pseudo-Measurement Technology task and the Non-Invasive Instrumentation Technology task, as part of the Systems Evaluation and Assessment Project of the Engineering of Complex Systems (ECS) Block. It also complements the research efforts of the System Design Synthesis and the Systems Reengineering Projects of the ECS Block.

The authors would like to thank their sponsor, the Office of Naval Technology, especially CDR Jane Van Fossen, USN (Ret.) and Elizabeth Wald. The authors would also like to thank all of the individuals who provided technical support to refine this report and Adrien Meskin for her editorial support.

Approved by:

C. A. Kalivretenos

C.A. KALIVRETENOS, Deputy Head
Underwater Systems Department

ABSTRACT

This report explores the evaluation and assessment techniques that may be applied to the different phases of the system life cycle--from requirements specification to implementation and reengineering. At each stage of the system life cycle, there are suitable techniques that may be applied to predict the behavior of the system, some more appropriate than others. Currently, however, there is no formal method for applying this evaluation information to the forward or reverse engineering process. Consequently, maximum use of the information available to the system developers rarely occurs. In addition, this report investigates the issues relevant to the integration of evaluation methods to the system development process.

CONTENTS

CONTENTS (Cont.)

ILLUSTRATIONS

CHAPTER 1

INTRODUCTION

The current state of practice of systems engineering is a fragmented process. The tools and techniques used to develop and analyze systems in one phase of the life cycle generally do not transition to the next phase; instead, as the design progresses or different aspects need to be studied, a new set of tools and techniques must be used. This process is inefficient, expensive, and may allow the designer to miss potential problems or alternative solutions. What is needed is a fully integrated process that allows the seamless transition between phases and analyses in the system development process.

This research effort explores three modeling and analysis approaches that will aid in the development of naval systems: dynamic performance modeling, rapid prototyping, and executable specification.

BACKGROUND

Navy systems belong to the class of Mission Critical Complex Real-time (MCCR) Systems. These systems share the following characteristics: very large and complex; real-time; parallel and distributed architectures; and high degree of information overload. They also have very strict requirements: they must exhibit an extremely long life cycle and must meet high reliability, availability, fault-tolerant, stringent security, survivability, and performance requirements. For satisfactory performance, these characteristics must be considered throughout the system life cycle--from the initial conceptual system design to the system life cycle management. These characteristics, which exist in all system resources (i.e., hardware, software, or humanware), complicate the system development process. Currently, the processes used for MCCR system design are inadequate to guarantee satisfactory performance with respect to these factors.

CURRENT SYSTEM DEVELOPMENT PRACTICE

Current systems are developed in a fragmented, ad hoc manner. Present methods can be viewed as in Figure 1-1, in which the techniques are splintered and disjointed. Knowledge gained and tools and techniques used at one stage cannot be easily passed on to the next stage, nor can results and design modifications be directly fed back to the previous stages. Today's methods are limited by an inability to trace requirements and thoroughly perform consistency checking. As a result of these deficiencies, redundant modifications and inconsistent models may pervade the development process. Additionally, the disjointedness of current methods make early error detection difficult. The later design errors are discovered, the more costly they are to solve; thus, it is important to be able to use modeling and analysis methods early in the design process in order to reduce system cost and improve performance.

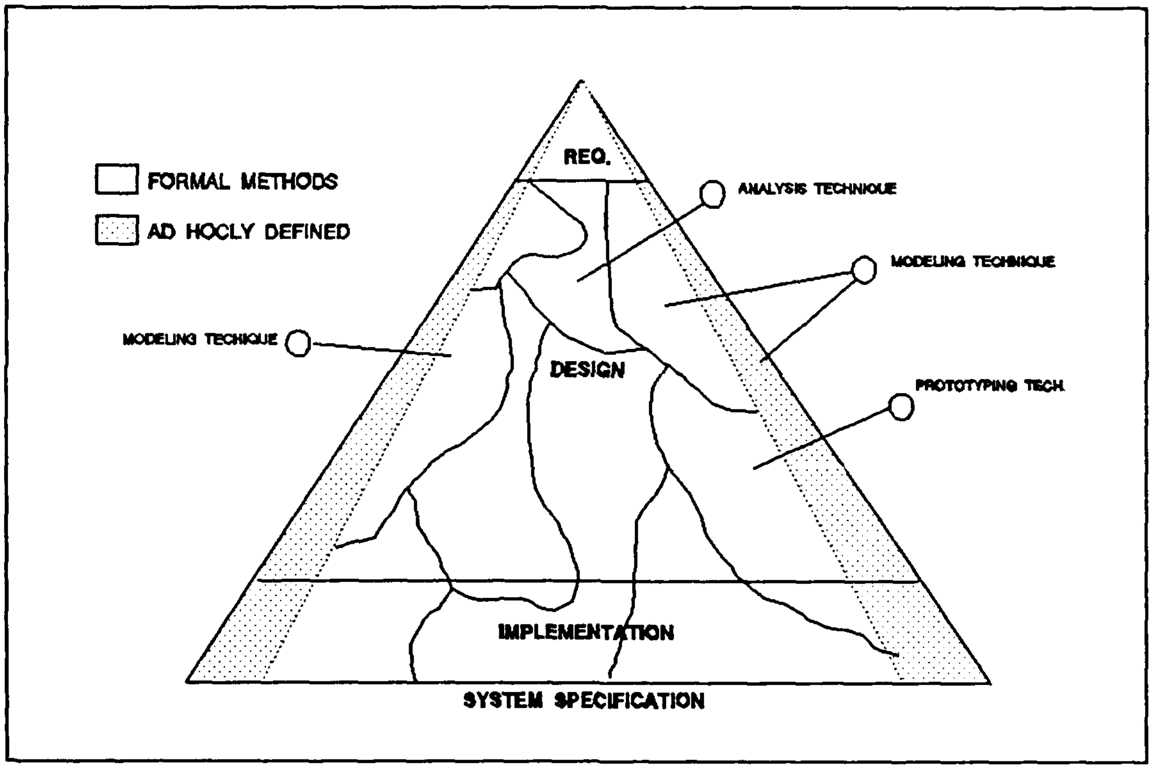

FIGURE 1-1. CURRENT SPECIFICATION PRACTICE

APPROACH

The development of an integrated, seamless path for system design depends on the advancement of three critical modeling techniques: dynamic performance modeling, executable specification, and rapid prototyping. Dynamic performance models may be classified as analytic or simulation. If the relationships which compose the system are relatively simple, a mathematical method (such as algebra, calculus, and probability theory) may be used to obtain an exact solution on questions of interest; models captured using such a method are called analytic models. Simulation uses the numerical logic models of a system to examine its behavior over time and under different conditions and scenarios. Simulation permits an arbitrary level of detail and complexity in modeling, and allows the study of the effects of simultaneous variations in several parameters. Executable specification is a formal representation of the system which captures the functional, behavioral, and implementational aspects. Finally, rapid prototyping is a representation of the system which bears closer resemblance to the final system than either dynamic performance modeling or executable specification. Rapid prototyping differs from the final implementation because in at least one area it does not meet the requirements.

CHAPTER 2

FUNDAMENTALS OF MODELING

Dynamic performance modeling, executable specification, and rapid prototyping share a common approach to system problems. Each technique must overcome the same shortfalls, and each uses a hierarchical modeling approach to solve problems.

PURPOSE OF A MODEL

A model is an abstraction of a system which is developed to allow analysis of certain aspects of the system. Models may capture the function, behavior, structure, and/or implementation of a system. Generally, a model is hierarchical: it begins as a simple statement of a proposed solution, and this statement is refined as more details are determined such as system specifications, component functions, software, hardware, and humanware. This model is then used to determine a solution, the ability of the solution to meet its requirements, the feasibility of such a solution, and the optimization of the solution.

Modeling, through trade-off analysis, allows quick searching of the solution space (i.e., a boundary formed by the constraints and requirements of the problem) and refinement of possible solutions. Current techniques restrict the designer to a limited solution starting space because of the need to provide cost effective and timely solutions. As a result, if the optimal solution is not in this limited space, it will never be reached. One of the goals of this research is to allow the designer to start with a very large initial solution space and to quickly reduce the solution field. This goal stems from the high cost of carrying each alternative solution further into the design process.

MODELING SHORTFALLS

Just as there is no single paradigm for modeling Discrete Event Dynamic Systems (DEDS), there is no single model that can capture all aspects of the system. Models are used to capture the part of the system that is of interest to the systems engineer. However, even with this limited scope, there are shortfalls with the models: the complexity of state dependent models tend to increase at an exponential level, mapping a model from one representation to another is extremely difficult, and the human aspect of the system cannot be easily modeled.

Complexity Limitations

Currently, although modeling techniques are capable of describing many systems, these techniques are usually constrained by complexity. For example, in state space models, the complexity increases exponentially with the number of states. Complexity limitations have kept modeling from being more widely applied. If methods were available which allowed transformation between modeling techniques, many of the complexity limitations could be overcome by using the strengths of a particular technique. This transformation between modeling techniques is not trivial since it is generally a many-to-many mapping. This many-to-many mapping makes traceability of the models and verification of the mapping extremely difficult.

Back Annotation Limitations

Systems engineering requires the ability to back annotate changes. As the systems develop and age, changes are made that will require updating the model. These changes require the ability to update the entire model, from the point where the change is made, back to the requirements specification. To be effective, back annotation and traceability demand that all development decisions and their rationale be formally documented. Thus, revisions in the system can be accurately reflected in all phases of the model.[1]

Humanware Modeling

Humanware modeling must be an integral part of the system engineering process in order to comprehensively assess the performance of a system, although it is not captured in most current modeling techniques and methodologies. "Humanware" implies the view of humans as resources which can be allocated functions and analyzed in the same manner as software and hardware. Humanware not only includes the response time and information handling capability of the actual humans who will interact with the system, but also the documentation, training, and other factors needed for the humans to be able to perform their functions. As an example, if the human was the bottleneck of a process but had not been modeled into the system, resources might be wasted trying to increase computational speed to no avail.

Operational Environment

Operational environment refers to the conditions under which the system (or subsystem) will actually be used. During system modeling the operational environment must be duplicated to exercise the model as closely as possible. The operational environment could differ considerably from the expected environment, the development environment, or the test environment. For example, the operational performance of the signal filter of a radar system for a high sea wave will be lower than expected if that system was evaluated for a low sea wave. The closer the model is to the operational environment, the better the prediction of the system's behavior.

HIERARCHICAL MODELING

Hierarchical modeling is the breakdown of a system model from the higher level system description to the lower level component description. It is a technique that is employed in the design of complex systems to maintain the complexity at a manageable level. System design, using a hierarchical process, begins with a high-level description of the total system and works down to a progressively more detailed description of individual system modules. By gradually refining the details of the system, complexity on each level of hierarchy of system design is maintained at a manageable scale. Initially, these descriptions of the system are independent of the implementation technology, because in the early stages of design, not tying the system under development to a specific implementation provides the freedom to choose from a larger solution field. At the earliest stages of the design process, the designer has not even determined which system functions should be implemented in hardware and which in software. Therefore, early system descriptions are more behavioral than structural; i.e. they emphasize the system's function (what it should do) and performance (how fast and how well it should do it) rather than focusing on structural and interconnection details (how it should do it).

The hierarchical breakdown from system to component level allows the prediction of behavior of the system, and identification and resolution of potential bottlenecks. If the system-level representation and analysis were to be skipped, system level performance information might be missed. A system's performance evaluation requires more than the aggregation of the performance of the individual components. Alternately, component level details provide a means to perform a detailed analysis of critical components that are isolated, and to seek alternative component choices for design trade-offs.

Part of the purpose of the hierarchical decomposition of a model is to define the boundary between the system and its environment and the boundaries among the components of the system. The boundaries must provide high cohesion and low coupling with the components. Cohesion refers to the level of association among component elements of a module; and coupling refers to the measure of the strength of interconnection among the modules. High cohesion and low coupling make system designing easier by reducing the interconnection between components.

CHAPTER 3

DYNAMIC PERFORMANCE MODELING

Dynamic performance modeling is used to model dynamic systems, those which either change in time or whose response changes in time; i.e., MCCR systems. The category of dynamic systems can be further divided into Continuous Variable Dynamic Systems (CVDS) and Discrete Event Dynamic Systems (DEDS). CVDS generally occur in nature and are well modeled using ordinary and partial differential equations. An example of CVDS is the behavior of the earth's orbit of the sun. The earth's behavior is a continuous motion described by Newton's law of gravity. In comparison, DEDS are generally man-made or are systems in which a human has intervened and as such, have very complex, event driven logic. An example of DEDS is the detection of a missile. A missile goes abruptly from undetected to detected without a gradual progression. In DEDS, the system changes only at discrete instants, takes value from a discrete set, and lasts a distinct period of time. Thus, a state and a state holding time can be used to characterize a DEDS system.

The difficulty in using DEDS is the complex nature of the systems and the lack of a common paradigm. Because most DEDS are man-made or man-intervened, they have extremely complex logic. This makes modeling very difficult. Additionally, although there have been numerous models proposed for their abstractions, a single paradigm is lacking for the modeling and analysis of DEDS. This is especially true for systems that have hard, real-time requirements and distributed and parallel architecture implementations.

USES OF DYNAMIC PERFORMANCE MODELS

Dynamic Performance Analysis is used to determine three basic responses of the system: steady-state, transient, and dynamic. Techniques for analyzing concurrent DEDS differ in their treatment of the parameter time.

Steady-State Analysis

Steady-state analysis is intended to determine the time average behavior of the systems with the average input from its environment. It is useful for analyzing the average stability of the system. As an example, steady-state analysis is used to determine the average number of the customers in a simple queue. However, the complexity involved in determining the accurate steady-state behavior of a system makes it difficult to obtain a closed form solution. To obtain this solution, many independent assumptions have to be made for a large network. These assumptions often limit the accuracy and usefulness of the solutions. To date, most models fall into the steady-state class. The problem with analyzing steady-state models is that the DEDS will either be over-designed and/or may not meet short but stressful scenarios. Thus, for the requirements of a hard real-time system, little value is gained from steady-state analysis.

Transient Analysis

Transient analysis uses the average inputs from an environment to model the system's behavior as a function of time. Because of the difficulty in obtaining a closed form solution even with many independent assumptions, simulation is most often used for transient analysis.

Dynamic Analysis

Dynamic analysis involves varying both system and environment parameters with respect to time. Thus, dynamic analysis is needed to model mission critical systems that may have capabilities changing over time (such as degraded modes of operation) and an environment changing over time (such as a transition from peace time scenario to hostile engagement). There is a shortage of techniques or tools for handling the dynamic behavior of such systems.

Solutions to the transient/dynamic situation must take a dual approach. This approach analyzes the system using operational environmental parameters and internal system structure. In the environmental approach, scenarios are developed and tested which are expected to stress the system; the scenarios are developed independent of the system construction. Alternatively, the system can be analyzed using the critical path approach. This type of analysis uses knowledge of the system structure to determine the test cases. Results depend largely on the success of the developer to find "proper" test cases. Unfortunately, it is very difficult to determine if a system has been fully tested; thus, unexpected performances could result. A combination of these approaches would allow the systems analyst to better understand the system; however, there is a substantial effect on analysis results and the prototype's performance due to the order and method with which these two approaches are applied.

TYPES OF DYNAMIC PERFORMANCE MODELS

Dynamic Performance models can be divided into two categories: analytic and simulation.

Analytic Models

Analytic models of computer systems are sets of mathematical equations whose independent variables (inputs) produce a single set of dependent variables (outputs). Analytic models give extremely fast, reasonably accurate, and acceptably detailed solutions to many problems.

Analytic modeling is most useful during the operational phase of a computer system's life cycle for testing the effect of relatively small changes to various components of the equipment configuration. It is also useful for estimating when bottlenecks will occur and which parts of the configuration will be the likely causes of these bottlenecks. Examples of applications for analytic modeling are complexity, difference equations, state space, queuing network, computer-based, min-max algebraic, and perturbation analysis models.

Although analytic techniques have existed for a long time, they have had limited use because it was believed that computer systems were too complex to be handled by such a simple approach. Oddly enough, this approach has proven most valuable in analyzing on-line, transaction-oriented systems which are most difficult to examine by using simulation or other analysis methods. Many simplifications are necessary to apply analytic models, but for the most part, these do not severely restrict the modeler nor do they limit the usefulness of the results.

Simulation Models

In contrast to analytic modeling, simulation modeling can be used for both simple systems and for complex systems which cannot be represented in closed form solutions. It tends to be more detailed than analytic. Simulation models mimic the system and its environment. They provide a large amount of flexibility because they may be executed using either real data collected from similar systems or from simulated data.

Comparison of Analytic Modeling and Simulation Analysis

Analytic modeling is most useful for the rapid development of a model and the quick assessment of alternative solutions. However, since analytic modeling lacks the ability to model detailed information, further analysis must be performed using simulation. Not only does simulation allow the inclusion of more details, but it also allows the use of elaborate scenarios as input. One of the major disadvantages of simulation is the extensive need for high speed and large memory computers. With both analytic and especially simulation models, it is very difficult to feed results and modifications back to the original system specifications.

A comparison of the two modeling techniques follows:

Analytic Techniques

* provide responsive evaluation of trade-offs
* provide inexpensive model execution
* allow application of performance optimization techniques
* allow thorough sensitivity analyses
* are limited in their level of detail by the ability of mathematical equations to accurately reflect the behavior of complex systems.

Simulation Techniques

* allow the incorporation of low-level system details
* have the potential for highly accurate performance predictions
* allow the detection of system logical design defects
* often are prohibitive in their time and cost for development

MODELING AND ANALYSIS TECHNIQUES

There is no single paradigm for the modeling of dynamic systems. The following is a classification of the dynamic models that abstract the system. Classification varies among different authors; therefore, the following classifications are not the only true taxonomy.

Queuing Theory

In queuing models, a queuing network of nodes is created to represent a client-server relationship. Within the network, nodes take the place of servers, and tokens take the place of clients. Nodes and tokens are given attributes to determine which tokens may be served by which nodes. Probability distributions are used to determine the rate at which tokens are introduced to the network and how quickly they are served by the nodes. Links and queues are established with specific lengths and types to hold waiting tokens. If simulation techniques are used, the network is time-stepped, and the states of the network are recorded for calculating the results. As the link capacities are reached, token flow delays are increased. This simulation is used to examine average queue lengths and delays, to determine the stability of the system, and to determine routing algorithms for reducing congestion. If analytic techniques are used, the probability functions are evaluated to find the solution.

One of the most important measures of a data network is the average delay required to deliver a packet from origin to destination. Furthermore, delay considerations strongly influence the choice and performance of network algorithms, such as routing and flow control.

Queuing theory is the primary methodological framework for analyzing network delay; i.e., the average delay required to deliver a packet from origin to destination. This delay strongly influences the choice and performance of network algorithms, such as routing and flow control. The use of queuing theory often requires simplifying assumptions since, unfortunately, more realistic assumptions make meaningful analysis extremely difficult. For this reason, it is sometimes impossible to obtain accurate quantitative delay predictions on the basis of queuing models. Nevertheless, these models often provide a basis for adequate delay approximations, as well as valuable results and worthwhile insights.

Graph Theoretic

The graph theoretic approach represents the system as a graph model, similar to a discrete event simulation. However, the inherent graph is examined using analytic approaches to determine the performance given different environment scenario attributes. The graph is typically collapsed into an equational representation in this approach. Though the collapsing of the graph is a complex and time consuming task, it needs to be done only once for each graph and is easily automated. Once in collapsed form, the system model can be analyzed quickly with different scenarios. This approach has been used for determining software bottlenecks and analyzing parallelization.

Petri Net

A Petri net is a graphical and mathematical modeling technique applicable to many systems. As a graphical tool, a Petri net can be used as a visual-communication aid similar to flow charts, block diagrams, and networks. In addition, tokens are used in these nets to simulate the dynamic and concurrent activities of the systems. As a mathematical tool, it is possible to set up state equations, algebraic equations, and other mathematical models governing the behavior of systems. Finite state machines or their state diagrams can be equivalently represented by a subclass of Petri nets.

The behavior of many systems can be described in terms of system states and their changes. In order to simulate the dynamic behavior of a system, a state or marking in a Petri net is changed according to the transition (firing) rule. There is no strict limit on the number tokens that are allowed on the nets; therefore, parallel activities or concurrency can be easily expressed in terms of Petri nets. One of the advantages of a Petri net is that it can be used to represent not only the flow of control but also the flow of data. Many abstraction techniques lack this ability. Dynamic performance models tend to emphasize control information and provide little opportunity to represent data flow and data transformation. Petri net simulation uses tokens to represent data. Tokens reside in places and pass from place to place via transitions, where they are processed. In a strict Petri net simulation, transitions occur independent of time. In common practice, Petri nets are generally enhanced by elements which introduce randomness, time dependency, and attributes to tokens and transitions. Design errors such as deadlocks and race conditions can be discovered through the use of Petri nets.

Petri nets is a promising technique for describing and studying information processing systems that are characterized as being concurrent, asynchronous, distributed, parallel, non-deterministic, and/or stochastic.

The major weakness of the Petri net is the complexity problem; i.e., Petri net based models tend to become too large for analysis even for a modest size system.

Communication protocols are another area in which Petri nets can be used to represent and specify essential features of a system. The liveliness and safeness properties of a Petri net are often used as correctness criteria in communication protocols.

One of the disadvantages of the original net was that it lacked too many control structures, such as not being able to represent timing information and stochastic properties in the nets. The following attributed Petri nets are some of the responses from the Petri net community invented to remedy the disadvantages.

<u>Time Petri Net</u>. The concept of time is not explicitly given in the original definition of Petri nets. However, for performance evaluation and scheduling problems of dynamic systems, it is (at present) necessary and useful to introduce time delays associated with transformation and/or places in their net models. For example, transition t_i can be disabled until the tokens for all the input process are in place for d_0, but not more than d_1, where d_0 and d_1 are some constants.

DETERMINISTIC TIMED PETRI NET. If the time delay associated with the transformation is deterministic, such models are called deterministic timed Petri nets.

STOCHASTIC TIMED PETRI NET. If the time delay associated with the transformation is stochastic, such models are called stochastic timed Petri nets. This is useful for modeling transaction systems, which are normally modeled with queuing networks. A few researchers are focusing on the possibility of transforming the Petri net model to a queuing network model to use the advantages of both models.

Stochastic Petri Net. A stochastic Petri net is a Petri net in which each transition is associated with an exponentially distributed random variable that expresses the delay from the enabling to the firing of the transition. In a case where several transitions are enabled, the transition that has the shortest delay will fire first.

Colored Petri Net. In colored Petri nets, each token has an attribute that distinguishes that "colored" token from another different "colored" token. Added attributes provide more capabilities to the Petri net models. One may use the attributes for a variety of reasons: to set up a priority scheme according to the colors, to have separate queue lengths for each color, to dictate how many and which kind of colored tokens will be removed from and added to the places, or any of the many other possible uses.

Computer-Based Models

Computer-based models are algebraically-based models which aim to capture the description of the trajectories of DEDS. These trajectories are described in terms of a small set of algebraic operations of state and events in very much the same way CVDS are succinctly described by differential and algebraic operations on functions of state and inputs.

Communicating Sequential Processes (CSP). CSP is one of several processes algebra developed for reasoning about concurrency, communication and distributed systems. Process is used as a mathematical abstraction of the interactions between a system and environment. The system is assembled with the processes; in the system, the components interact with each other and with their external environment. A CSP communicates with its environment through named communication channels. The behavior of the process is recorded as a trace of actions in which it engages. Therefore, a trace is an observation of process execution. CSP notation allows the description of processes using a variety of process constructors. This approach provides a secure mathematical foundation for avoidance of errors such as divergence, deadlock and nondeterminism.

CHAPTER 4

EXECUTABLE SPECIFICATION

Executable specification, in its most fundamental definition, captures the functional and behavioral descriptions of the system. Like most terminologies, it has been used indiscriminately to describe anything from simple simulation to prototyping. Executable specification, if robust and inclusive of all views of the system, will serve as the design specification and single representation of the system. If other views or representations are needed, they can be generated from this "global" representation. With only one representation of the system serving both system modeling and design specification, all modifications made during system modeling would automatically be reflected back in the design specification.

In the streamlined process, executable specification replaces several descriptions of the system (requirements specification, design specification and implementation specification); therefore, it must serve many roles:

* It must serve as a specification of the functional requirements for the system.

* It must be validated: it must answer questions such as, "Is it internally consistent?" "Is it consistent with its nonfunctional requirements?" "Does the behavior of the specification agree with scenarios?"

* It must lead to an implementation that meets the systems's resource and performance requirements.

Executable specification should combine both data manipulation and control in a formal representation, since analytic modeling techniques emphasize control but not data, and data flow diagrams emphasize data but not control. Executable specification should provide a capability to formalize all functional properties of a system.

In its purest form, executable specification should allow any other representation to be generated from the original representation; however, this is currently not possible. It is generally believed that no specification will ever be able to play all these roles for all systems, largely due to the fact that there is no one-to-one mapping among models captured using different techniques. A global representation of a system, as shown in Figure 4-1, is useful, because it would allow designers to extract the specific modeling representation which they need. Executable specification is amenable to top-down design methodology, where the functional specification is the primary source of guidance in design processes.

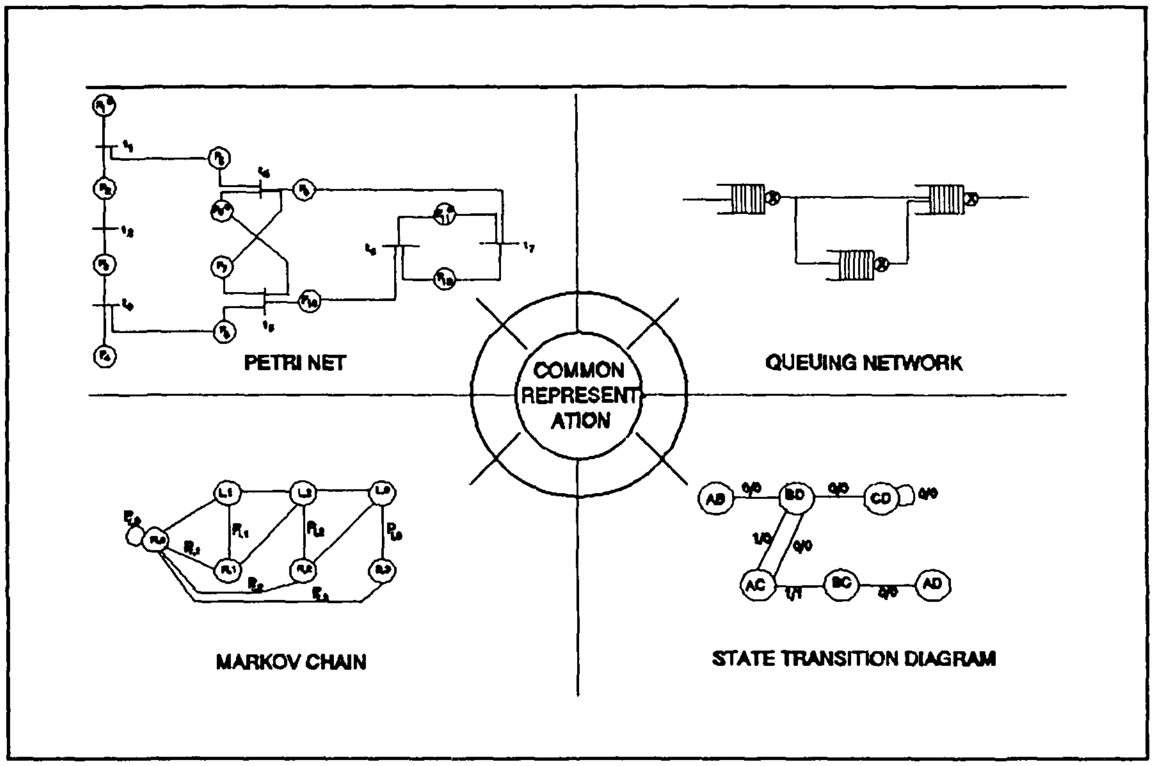

FIGURE 4-1. COMMON PERFORMANCE MODEL REPRESENTATIONS

An executable specification may be used for the evaluation of functional and behavioral correctness and consistency. Executable specification is being considered as a promising approach to rapid prototyping software.

In this approach, physical resources are modeled using resource models. The mapping between the logical model and the resource model allocates the functions to the physical resources. There are two levels of agenda in this mapping. On one level, mapping is performed to obtain a reasonably acceptable implementation of the system. On the second level, mapping is performed to obtain an optimal implementation of the system. The criteria used in determining the optimality is given by the metrics obtained from Reference 2. The priority level of each criteria will depend on each system under development and the customer.

There are many companies claiming that their product performs executable specification, but in the context described above, PAISLey, Statemate and Structured Analysis with Real-Time Extension actually represent prototypical executable specifications.

COMPONENTS OF EXECUTABLE SPECIFICATION

If executable specification provides a "global" model capable of representing information for all views and all models, it is necessary to determine which views and models constitute an adequate representation of the system. The following models and views have been chosen as adequate to represent Navy MCCR systems.[3]

Logical Model

The logical model consists of two parts: the functional view, a model which focuses on defining system interaction, and the behavioral view, a model which describes required system behavior. Both views are implementation-independent. However, both must account for the implementation technology used in the environment, which is not a part of the system at this level.

The functional view provides a conceptual approach by identifying a hierarchy of activities, complete with the details of the data items and control signals. However, the functional view does not specify dynamics: it does not state when activities will be activated, whether or not they will terminate on their own, nor whether they can be carried out in parallel or must be executed sequentially. The functional view only specifies that data can flow but not whether and when it will flow. The functional view decomposes the system into activities and possible flows of information, but it says little about how these activities and their associated inputs and outputs are controlled during the continued behavior of the system.

The behavioral view is responsible for specifying control and timing. This is achieved by allowing a control activity to be present on each level of the activity hierarchy. Each control activity controls its own particular level. It is these controllers that are responsible for specifying how, when, and why things happen as the system reacts over time. The techniques and symbology for the behavioral representation of the system vary from control graphs to decision tables to flow charts. The behavioral view must have a means of representing timing relationships along with control information.

Resource Model

The resource model is broken down into two levels: global resource model and internal resource model. The global resource model consists of both the internal and the external resource models. An internal resource model contains all of the hardware and software modules, and hardware architectures that are candidates for the mapping of the functions of the logical model.

External/Environmental Model

Once the system is defined, everything outside of that system is considered external. Because the external world (environment in which the system will operate) interacts with the system, and thus, affects system performance, the designer must be able to capture those aspects of the external world which impact the system.

External conditions which affect the systems are called scenarios. A system's performance under the worst case scenario provides more information about the system's ability to operate in the real world than that of under best case scenario. Therefore, worst case scenarios must be determined and used in the assessment of system performance.

Implementation Model

The implementation model is the mapping of the logical model to the resource model. It includes resource descriptions and a mapping which relates the functions, data flows and control from the logical model directly to the resources. The task of implementing a real-time system can be long and complex. Although optimal allocation of resources must be considered,[4] this project currently only emphasizes viability.

RESOURCE MAPPING

Resource mapping of the activities and controls in the logical model to a physical model is crucial to system performance. Until this phase, all the constraints assumed were based on the designers' experience and estimates. Now through designer ingenuity, specifications must be met in spite of actual physical constraints imposed by technology. Finding the optimal resource allocation/mapping is a very complex issue; however, for this task, the objective of the resource mapping is to obtain a viable solution. As long as the constraints of the requirements are met, the mapping is considered as a candidate implementation model.

ANALYSIS OF EXECUTABLE SPECIFICATION

The analysis of executable specification will be immensely more complicated because of the information that it contains. Executable specification contains so much information because it has to serve as a single global representation of the system and serve many roles for many people. This complexity requires a better method to check that the information is consistent from one stage to another and the specifications are done correctly and completely.

Completeness

Completeness is a loosely defined term that describes the state of the behavioral specification of the system. It is very difficult to declare a system functionally complete, in the sense that a specification captures all the functionality and the behavior of the system. However, it is not too difficult to check that the specification given contains all the information required for a specific methodology.

Correctness

Functional correctness checking is the process of determining that the logical model of system function, data, and control is free from functional conflicts and shortfalls. An example of functional conflict on a communications model would be a conflict with routing messages across links; and a functional shortfall would be the inability to process multiple destinations for a single link.

Executable specifications should support functional correctness checking with symbolic simulation. They should verify that subsystems transition to the appropriate states when presented with various stimulus events and that transitions cause the appropriate changes in the activation and deactivation of processes.

CHAPTER 5

PROTOTYPING

Prototypes are classified as models because they are abstractions of the system. Prototypes vary from containing only some functionality of the final configuration to being the production configuration in all aspects, except that they have not been fully tested. The assessment of the system up to this point has been performed with the model mimicking the system and providing a certain level of confidence that all the requirements have been met. However, there is a need to verify concepts and design configurations using the actual components. In essence, simulations mimic and prototypes do. Prototypes are also used to validate assumptions about the system and to test critical components, functions, performance and system feasibility. The measurements performed with the prototypes can be used to calibrate the model and validate that the requirements specification have been met.

In all cases, the usefulness of prototyping decreases with the increase in the time required to develop it. Even though the development cycle of large systems is long, the incorrect use or development of prototypes increases the time required. Rapid prototyping is a term used to define a method of prototyping that provides a means for a short development time. Since complex systems require prototyping be performed quickly, the remainder of this section will address techniques applicable to rapid prototyping.

By adding features of implementation, prototyping comes closer to reality than simulation. Software and hardware designs in the prototype can be used directly in the real network. If the prototype has an adequate monitoring and control system, data from experiments performed on the prototype can help tune design parameters and identify bottlenecks in the real network. The collected data and information must be abstracted to back annotate the system.

PROTOTYPING TECHNIQUES

Techniques of prototyping vary. As Table 5-1 shows, many of these techniques cross into other modeling domains (e.g., performance models and executable specification).[5]

TABLE 5-1. SOFTWARE PROTOTYPING, FORMAL METHODS AND VDM

TECHNIQUES	DOMAIN	ADVANTAGE	DISADVANTAGE
EXECUTABLE SPECS	FUNCTIONALITY	CONCISE AND PRODUCTIVE	NOT ALL SPECS ARE EXECUTABLE
REUSABLE MODULES	GENERAL	VERY PRODUCTIVE	INITIALLY EXPENSIVE
SIMULATION	GENERAL	EARLY APPLICATION	NO GENERAL SUPPORT TOOLS
VERY HIGH LEVEL LANGUAGE (VHLL)	LANGUAGE DEPENDENT	PRODUCTIVE	OFTEN CRYPTIC
AHLL	VERY RESTRICTED	VERY PRODUCTIVE	VERY APPLICATION DEPENDENT
STATE TRANSITION DIAGRAM (STD)	INTERACTION	GRAPHICAL	TEXTUAL NOTATION OFTEN CRYPTIC
TOOL-SETS	TOOL DEPENDENT	VERY PRODUCTIVE	INCOHERENT
FORMAL GRAMMARS	CERTAIN INTERACTIONS	CONCISE	INFLEXIBLE
FUNCTIONAL PL	FUNCTIONALITY	CONCISE	OFTEN INFLEXIBLE

There are three basic categories of rapid prototyping: throw-away, incremental, and evolutionary. Typically, prototypes service many different purposes in the development cycle; therefore, seldom will a prototype only belong to one category.

Throw-Away Prototyping

Throw-away prototypes help define initial specifications. These prototypes are used in the design phase as tools for exploring and evaluating the appropriateness and feasibility of alternative designs. During the testing of a developing system, a prototype can be used as a comparator that evaluates the correctness of the test results.

Throw-away prototypes are usually quick mock-ups and are typically not used in later generations of the model. The need for rapid development is greatest for throw-away prototyping. Since the prototype is to be used for a limited period, quality factors such as efficiency, structure, maintainability, full error handling, and documentation are typically of little relevance.

Evolutionary Prototyping

Evolutionary prototypes, in contrast to throw-away prototypes, evolve for use in later generations of models. This prototyping is needed because systems, once installed, evolve steadily, thus, invalidating some of their original requirements. The purpose of the evolutionary approach is to rapidly introduce a system and to allow changes that become evident from using the system.

Evolutionary prototyping is the most powerful way of coping with change. This approach requires a system to be designed in such a manner that the introduction of change does not lead to severe problems.

Incremental Prototyping

In incremental prototyping, the system is built one section at a time. A complete system is designed, and then modules are implemented and added sequentially. Incremental prototyping has often been used synonymously with evolutionary prototyping; however, there is a significant difference between the two. Incremental prototyping is based on one overall design; evolutionary prototyping evolves the design continuously. As with evolutionary prototyping, the system grows gradually, but in a considerably less dynamic way. Incremental prototyping provides less scope for adaption than evolutionary prototyping, but it has the advantage of being easier to control and manage.

ANALYSIS OF PROTOTYPES

Assessment of prototypes depends on the implementation method. Prototypes may be realized and developed through several different methods: emulation of the hardware on an existing architecture, implementation of a physical system, and simulation of a system. Once the prototype has been developed, the behavior of the prototyped system must be measured and evaluated. This task requires measurement techniques and control and observation systems.

Prototyping provides an opportunity to measure the system under its operating condition. There are three techniques for measuring a system: noninvasive, minimally invasive, and invasive.[6] Noninvasive instrumentation techniques measure the system without interfering with the regular operation of the system. Minimally-invasive instrumentation techniques require at most 5 percent of the operating overhead for maintaining and supporting instrumentation and data collection. Invasive instrumentation techniques require more than 5 percent of the operating overhead of the instrumentation. Therefore, the data collected from invasive instrumentation would not necessarily correspond to actual data generated under normal operating conditions. The ability to use these different techniques depends on the development method for the prototype.

Regardless of the instrumentation technique used, the data collected must be restructured for the user's understanding and for back annotation to the design and requirements specification. Users must be given data in a format which is easy to understand, so they can observe and control the behavior of the prototype and extract the required information from the observations. As with any modeling and evaluation technique, the information from measurement and analysis of the prototype is used to validate, modify, and optimize the system.

PROTOTYPE DEVELOPMENT ENVIRONMENTS

Two key requirements of a prototype development environment are that they provide both a means to maintain the complexity of prototyping to a manageable level and a means to prototype rapidly. Maintaining a manageable level of complexity in the prototype gives the developers the capability to make the changes and observe the ramifications of the changes. An environment must be amenable to rapid prototype development because of the short duration of time allotted to prototyping. Three important components of a rapid prototype development environment include a user interface, a component library, and a control and observation structure.

User Interface

A user friendly interface is important in prototyping because it minimizes problems associated with the complexity of prototyping and analysis and measurement of data. In addition, it reduces the effort and memorization required for the reuse of designed components in the library; thus, it allows easy modification of prototypes.

Component Library

Development using reusable components requires an extensive component library. The hardware and software modules stored in the library have to be either supplied by the manufacturer or hand coded into the library by the system designer. The advent of the hardware description languages, VHDL and Verilog, has made reusable component prototyping a viable technique for developing a system.

Control and Observation Structure

The control and observation system manages the hardware modules, controls the evaluation session, and monitors the behavior of the prototyping environment. It allows the user to specify the system and evaluate the design.

Once the prototype is implemented, the control and observation structure allows prototype developers to observe the behavior of the system, and the ramifications of the changes to the system.

EXAMPLES OF PROTOTYPE DEVELOPMENT ENVIRONMENTS

Prototype development environments for software have existed for a long time. Only recently have environments addressed systems prototyping. The following are three examples of prototype development environments. The Computer-Aided Prototyping System (CAPS) is based on a reusable component software prototyping environment. The Integrated Design Automation System (IDAS) takes software as the requirements specification and provides an architecture that meets the requirements. Finally, the Programmable Network Prototyping System (PNPS) provides a quick and easy method of specifying network protocols and emulating the architecture in an existing architecture.

Computer-Aided Prototyping System (CAPS)

CAPS is a software prototyping tool being developed by Dr. Luqi of the Naval Postgraduate School. CAPS uses an integrated approach to prototyping that combines a computational model tailored for real-time systems, a high-level prototyping language (PSDL), a systematic design method for rapid prototype construction, and an automated prototyping environment that lets designers effectively use a base of reusable software components.

The computational model prevents hidden interactions among system components and encourages designs with good module independence. The language supports the model and combines it with a powerful set of data control abstractions to make it easy to describe systems at a high level. The automated environment relies on a software-base-management system for retrieving and adapting reusable components that are written in Ada.[7]

Integrated Design Automation System (IDAS)

IDAS, a tool set developed by JRS Research Laboratories, is geared toward embedded computer applications which demand high performance. The quality of computers designed for these applications is measured by the execution speed of the problem set on the machine. Design inadequacies are measured by the difference between actual performance and optimum performance. IDAS automatically derives relevant information about hardware and software designs while mapping Ada programs onto machines described in VHDL. It synthesizes machines described in VHDL from specifications expressed as Ada programs, retargets microcode compiler tools from a VHDL machine description, and guides users in rapidly evaluating numerous design alternatives and comparing the alternative approaches objectively and quantitatively.[8]

Programmable Network Prototyping System (PNPS)

PNPS uses a collection of reusable hardware modules to implement generic communications functions such as transmission, reception, signal propagation, and pattern matching. The modules are interconnected and configured to emulate a variety of communication networks whose behavior can be monitored under different workloads. The user specifies a network as a set of interacting components using available software tools. These tools are accessible within the prototyping environment that includes a control system for configuring the hardware modules and interconnecting them according to the component specifications. As with any other reusable prototyping system, the previously used components are stored in a component library.[9]

CHAPTER 6

APPLYING SYSTEM MODELS

Models are abstracted to obtain information about a system. The information is obtained by applying a workload and test vectors to the system. The response of the model to these inputs provides useful information about the system's performance, and its functional and nonfunctional capabilities.

WORKLOAD

The workload consists of the processing requests made to a computer system. An increased workload which occurs under stressful operating conditions, such as a hostile attack on a Navy ship, degrades system performance through increased demands on system resources. Because of workload effect, the performance of a system must be determined in the context of the demand on the system. Thus, the designer must be able to characterize workloads as close as possible to the real workload under all scenarios and conditions in order to fully assess the performance of the system.

Generally, the workloads of non-MCCR computer systems have certain statistical properties that do not change over long periods of time. These workloads are characterized by the distribution of demands made on the individual system's resources. For example, the workloads on analytical models are usually Monte Carlo type, in which the workload is characterized by a stochastic representation. This allows a closed form solution to be obtained. This type of characterization is useful for MCCR systems for steady-state analysis but not for transient and dynamic analysis.

For transient and dynamic analysis, trace-driven types of workload characterizations based on actual workloads or simulated workloads are needed. Simulation models use both statistical- and trace-driven workload characterizations.

Component Versus System Level Workload Characterization

Performance comparison between different system configurations should be calculated for the same workload because of the dependence between performance and workload. However, at the component level, performance can be evaluated in either a stand alone or a system environment.

To evaluate the performance of a particular component or system configuration, the designer must provide data concerning the performance of those modules. The evaluation output will show the overall performance and the bottlenecks that limit the performance. The designer can identify the modules most affecting system performance and optimize these modules during the design phase, while using a more conservative design for the remaining modules.

Limitations Of The Workload Characterization

Due to contentions between processors for shared resources, the systems' performance for a given workload and the increased performance gained from adding extra resources (processors, busses) are hard to quantify.

Another current limitation of the workload charcterization is the effect of static and dynamic reallocation techniques. This is a difficult problem in modeling. A task may not run on the same resources every time it is executed, or even during execution. If it is known a priori that a task will be shifted from one set of resources to another due to present system state, then static reallocation takes place. If, on the other hand, resource allocation is determined at every moment according to some algorithm which takes into account such system parameters as task load, and it is impossible to know exactly which resources will be used by any given task, then dynamic reallocation results. Current commercial simulation tools do not address either type of reallocation.

TEST VECTORS

One of the key elements of the system design process is the development of test vectors (scenarios). These test vectors provide specific inputs to the system that can be used to verify the systems' ability to perform its intended function within the required time constraint. Each stage of the system design process, therefore, consists not only of behavioral descriptions but also of the necessary validation test vectors.

ANALYSIS

Analyses of the models provide a means to assess the "goodness" of the design and allow the designer to seek alternative design choices. Analyses provide data based on the criteria provided by metrics. Alternative design choices are derived from this information and the requirements specification.

Types of analysis include deadlock, starvation, response times, resource utilization, sensitivity, trade-off, fault tolerance, and graceful degradation.

Deadlock

Deadlock is a situation which occurs in resource allocation systems, distributed systems, and communication networks. These systems have two or more processes which are in a simultaneous wait state, each waiting for one of the others to release a resource before proceeding. The same situation may occur in a database system where locking of data is used as a concurrency control mechanism.

A classic example of deadlock is a case of five scientists sitting at a round table waiting for dinner. On the left-hand side of each scientist is a fork; therefore, there are five scientists and five forks. In order for a scientist to commence eating, that scientist must have two forks; one in the left-hand and one in the right hand. However, when everyone tries to grab a fork, each scientist can grab only one fork, and no one eats.

There are three methods for dealing with the deadlock problem: prevention, avoidance, and detection combined with recovery. Prevention and avoidance methods ensure that the system will never enter a deadlock state. However, these techniques are not very mature and have not been shown to work for complex systems. Even if these techniques were to become effective for complex systems, they would not prevent existing systems from having deadlocks.

Starvation

Starvation is a situation in which messages have been sent that cannot be accepted by the destination module because it is in a nonterminating atomic transaction. A system is starvation free if there are no nonterminating atomic transactions.
An atomic transaction can fail to terminate because it contains an infinite number of events, it enters deadlock, or because the response to some event has an infinite delay. The first two cases are properties of a specification whose absence can be checked. The last case can arise only for implementations that do not conform to their specifications, because every response must be produced in a finite time by a correct implementation.

Response Times

In real time systems, the response times of a system are critical to the conformance of the system to its requirement specifications and to the survivability of the system. The response time specification defines the limits on the response time allowed between events occurring at the system input terminals and the resulting events exiting at the system output terminal. Typically, the system is embedded in a vehicle or in a larger system, so size, processing capacity, and memory capacity are limited. Achieving these critical timing requirements under these constraints becomes very difficult and is a major challenge for the designer.

Resource Utilization Analysis

Resource utilization analysis provides the basis for the allocation of resources for the optimal performance of the system. For example, it is better to assign, in most non-time critical cases, a faster processor to handle a heavier load than to assign a slower processor to the task. This allocation ensures that more processing will be performed in a given time. In more complex systems, these allocation strategies become very difficult because of the interdependencies of the functions and data, and the increased number of processors and alternative allocation strategies.

Sensitivity Analysis

The sensitivity of the model indicates the variation of the outputs to changes in inputs and model parameters. Sensitivity analysis, which measures the sensitivity of the model, provides useful information about how much a model parameter or input can change before a critical response results. This analysis allows one to limit the number of parameters which must be analyzed. (It is impossible for a designer to analyze all the parameters, inputs, and scenarios of a system; therefore, the designer has to determine which parameters and scenarios are important to the development of the system.) For example, if the system can withstand a wide range of temperatures before the performance or reliability changes, then temperature analysis is not as useful as analysis of another parameter, which, with the slightest change, greatly affects the output.

Trade-Off Analysis

Trade-off analysis allows designers to see the result of exchanging the requirements of one constraint for another. For example, by reducing the redundant copies of data in the memory, the reliability of the system can be traded off with the performance. Thus, reliability of a system will be degraded at the expense of the gain in the performance. This analysis works at every level of design, starting from the requirements specification to the implementation. It may be as simple as comparing two design choices or as complicated as looking at a large solution space within the constraints and trying to search for optimal solutions.

Fault-Tolerance and Graceful Degradation Analysis

For Navy systems, fault tolerance and graceful degradation of the system is as critical as timing requirements. Fault-tolerance analysis predicts the system's ability to tolerate and avoid faults. Thus, this analysis measures the performance of the fault-tolerance and avoidance strategies that are employed in the development cycle of systems.[10] Degradation analysis predicts the system's ability to degrade gracefully and recover from failure. For example, the system's ability to perform functions and operate within requirements specifications with 70 percent processors in operation and the other 30 percent in failure depends heavily on the system's ability to reallocate functions from failed processors to active processors. In most of the system model development environments, the fault tolerance issue is not addressed until the implementation model building stage.

Current use of the reliability and availability modeling techniques and tools are used to assess the system's availability and reliability as a stand alone capability. These techniques and tools need to be an integral part of the design methodology. Fault-tolerance and graceful degradation strategies need to be applied as early in the development cycle as possible, and the modeling and analysis techniques need to provide feedback information to calibrate parameters, and perform trade-off analyses.

CHAPTER 7

CONCLUSION

Most of the problems in modeling and analysis technology exist, not because of a lack of tools or methodology, but because of a lack of integration and a failure to address system level issues. There are many tools that address some system problems or some phases of system development. Although these tools alone are not enough to develop all systems, when integrated, they could provide many capabilities for system development. Currently, this integration has been lacking, including the consistent usage of terms such as modeling, executable specification, and prototyping.

Through the improvement of current methods, more alternatives could be rapidly explored leading to better solutions. As increased emphasis is placed on reducing system development costs and improving system performance, an integrated, seamless path will be required as shown in Figure 7-1. Instead of separate tools and techniques for each stage of development or aspect of design, there would be an integrated route between phases. This route would allow interaction among all phases including allowing information to be passed forward and to be fed back to previous stages. Thus, modifications made later on in the design cycle could be fed back to permit the designer to assess their impact on the entire system. Additionally, future design techniques and tools must include requirements traceability and consistency checking. Dynamic performance evaluation models, executable specification, and rapid prototyping combined will integrate the fragmented methodologies, techniques, and tools that have been used in the system life cycle and reduce the costly development cycle.

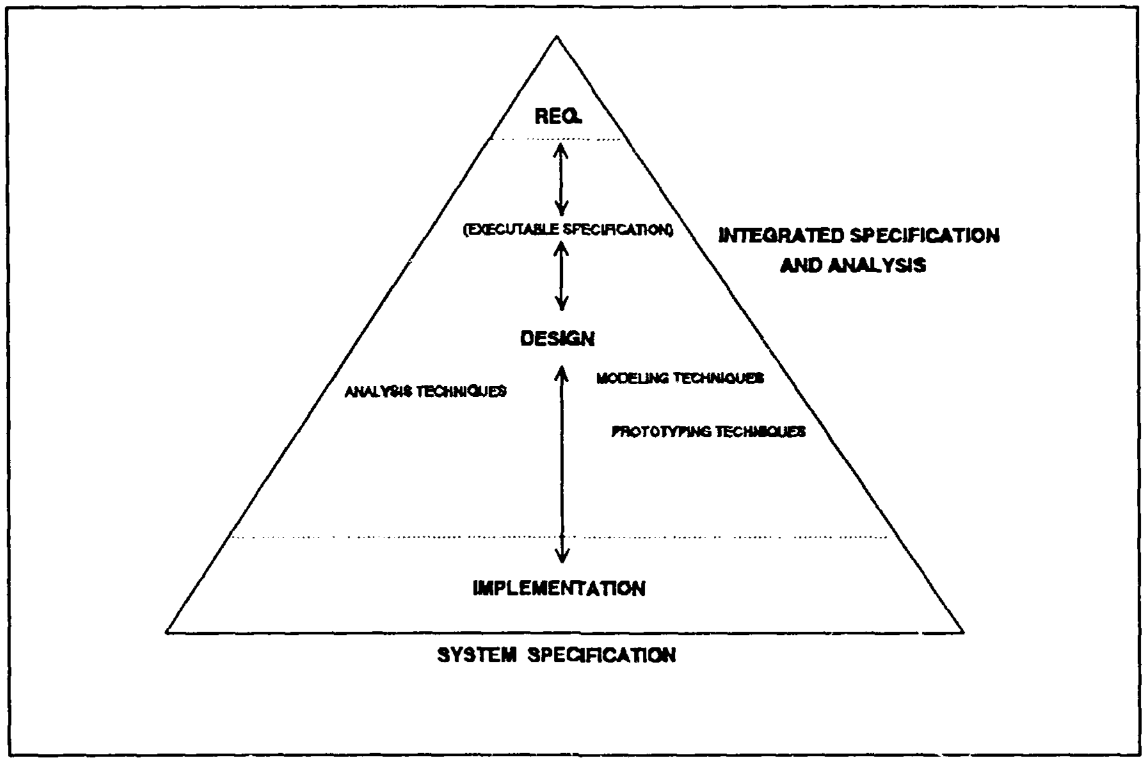

FIGURE 7-1. SPECIFICATION GOAL

REFERENCES

1. Edwards, M. and Howell, S., A Methodology for Systems Requirements Specification and Traceability for Large Real-Time Complex Systems, NAVSWC TR 91-584, Sep 1991, NAVSWC, Silver Spring, MD.

2. Farr, W., NSWC TR 82-171, A Survey of Software Reliability Modeling and Estimation, Sep 1983, NAVSWC, Dahlgren, VA.

3. Hoang, N.D., Howell, S.L., and Karangelen, N., Mission Critical System Development: Design Views and their Integration, NAVSWC TR 91-586, Oct 1991, NAVSWC, Silver Spring, MD.

4. Nguyen, C.M., Howell, S.L., and Hwang, P.Q., Expert Design Advisor, NAVSWC TR 90-46, Oct 1990, NAVSWC, Silver Spring, MD.

5. Murphy, K., Instrumentation Techniques for Uni-Processor and Distributed Systems, NAVSWC TR 91-596, Oct 1991, NAVSWC, Silver Spring, MD.

6. Hekmatpour, S. and Ince, D., Software Prototyping, Formal Methods and VDM, Addison-Wesley, 1988.

7. Luqi and Berzins, V., "Rapidly Prototyping Real-Time Systems," IEEE Computer, Sep 1988.

8. JRS Research Lab, "Integrated Design Automation System (IDAS)," Jun 1988.

9. Cieslak, R., et al., "The Programmable Network Prototyping System," IEEE Computer, May 1989.

10. Howell, S.L. and Wallenberger, A.P., Real-Time Dependable System Design, NAVSWC TR 91-590, Oct 1991, NAVSWC, Silver Spring, MD.

BIBLIOGRAPHY

Bertsekas, D. and Gallager, R., Data Networks, Englewood Cliffs, 1987.

Blanchard, B.S. and Fabrycky, W.J., Systems Engineering and Analysis, Prentice Hall, 1990.

Cieslak, R., et al., "The Programmable Network Prototyping System," IEEE Computer, May 1989.

Edwards, M. and Howell, S., A Methodology for Systems Requirements Specification and Traceability for Large Real-Time Complex Systems, NAVSWC TR 91-584, Sep 1991, NAVSWC, Silver Spring, MD.

Farr, W., NSWC TR 82-171, A Survey of Software Reliability Modeling and Estimation, Sep 1983, NAVSWC, Dahlgren, VA.

Hekmatpour, S. and Ince, D., Software Prototyping, Formal Methods and VDM, Addison-Wesley, 1988.

Ho, Y.C., "Dynamics of Discrete Systems," Proceedings of the IEEE, Vol. 77, No. 1, Jan 1989.

Hoang, N.D., Howell, S.L., and Karangelen, N., Mission Critical System Development: Design Views and their Integration, NAVSWC TR 91-586, Oct 1991, NAVSWC, Silver Spring, MD.

Howell, S.L. and Wallenberger, A.P., Real-Time Dependable System Design, NAVSWC TR 91-590, Oct 1991, NAVSWC, Silver Spring, MD.

Jain, P.P., "Architectural Models are Key to System-Level Design," Electronic Design, 28 Mar 1991.

JRS Research Lab, "Integrated Design Automation System (IDAS)," Jun 1988.

Karimi, J. and Konsynski, B.R., " An Automated Software Design Assistant," IEEE Transactions on Software Engineering, Vol. 14, No. 2, Feb 1988.

Luqi, "A Computer Aided Prototyping System," IEEE Software, Vol.5, Mar 1988.

Luqi and Berzins, V., "Rapidly Prototyping Real-Time Systems," IEEE Computer, Sep 1988.

BIBLIOGRAPHY (Cont.)

Murphy, K., Instrumentation Techniques for Uni-Processor and Distributed Systems, NAVSWC TR 91-596, Oct 1991, NAVSWC, Silver Spring, MD.

Nguyen, C.M., Howell, S.L., and Hwang, P.Q., Expert Design Advisor, NAVSWC TR 90-46, Oct 1990, NAVSWC, Silver Spring, MD.

Roman, G.C., Stucki, M.J., Ball, W.E., and Gillett, W.D., "A Total System Design Methodology," IEEE Computer, May 1984.

Svovodova, L., Computer Performance Measurement and Evaluation Methods: Analysis and Application (Computer Design and Architectures Series), Elsevier, 1976.

DISTRIBUTION

	Copies
Defense Technical Information Center	
Cameron Station	
Alexandria, VA 22304-6145	12
Library of Congress	
Attn: Gift and Exchange Division	4
Washington, DC 20540	
Office of Naval Technology	
Attn: Code 227	
(Elizabeth Wald)	1
(Cmdr. Gracie Thompson)	1
800 N. Quincy Street	
Arlington, VA 22217-5000	
Center for Naval Analyses	
4401 Fort Avenue	
P.O. Box 16268	
Alexandria, VA 22302-0268	2
Naval Research Laboratory	
Attn: Code 5543	
(Catherine Meadows)	1
Washington, DC 20375	
Naval Postgraduate School	
Attn: (Dr. Luqi)	1
Computer Science Department	
Monterey, CA 93943	
JRS Research Laboratories Inc	
Attn: Erwin Warshawsky	
1036 W. Taft Avenue	1
Orange, CA 92665-4121	
Advanced Technology & Research Corp.	
Attn: Adrien J. Meskin	5
George Stathopoulos	1
14900 Sweitzer Lane	
Laurel, MD 20707	

	Copies
Trident Systems Inc.	
Attn: Nicholas Karangelan	1
10201 Lee Highway	
Suite 300	
Fairfax, VA 22030	
Internal Distribution:	
D4	10
E231	2
E232	3
E342 (GIDEP)	1
F01	1
G	1
G42 (C. Yeh)	1
K02	1
K52 (W. Farr)	1
N30 (H. Crisp)	10
R44 (H. Szu)	1
U	1
U02	1
U042	1
U10	1
U20	1
U23	1
U25	1
U30	1
U31	1
U33	1
U302 (P. Hwang)	20
U33 (D. Choi)	15
U33 (M. Edwards)	1
U33 (N. Hoang)	1
U33 (S. Howell)	10
U33 (M. Jenkins)	1
U33 (T. Moore)	1
U33 (C. Nguyen)	1
U33 (H. Roth)	1
U33 (M. Trinh)	1
U40	1

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE 2 February 1992	3. REPORT TYPE AND DATES COVERED

4. TITLE AND SUBTITLE Systems Modeling	5. FUNDING NUMBERS PE - 0602234N PR - RS34P12 TA - Task 1
6. AUTHOR(S) Dong Choi, Janet Youngblood, Steve Howell, and Phil Hwang	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Surface Warfare Center (Code U33) 10901 New Hampshire Avenue Silver Spring, MD 20903-5000	8. PERFORMING ORGANIZATION REPORT NUMBER NAVSWC TR 91-592
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report explores the evaluation and assessment techniques that may be applied to the different phases of the system life cycle – from requirements specification to implementation and reengineering. At each stage of the system life cycle, there are suitable techniques that may be applied to predict the behavior of the system, some more appropriate than others. Currently, however, there is no formal method for applying this evaluation information to the forward or reverse engineering process. Consequently, maximum use of the information available to the system developers rarely occurs. In addition, this report investigates the issues relevant to the integration of evaluation methods to the system development

14. SUBJECT TERMS systems modeling; executable specification; rapid prototyping; requirements specification; discrete event dynamic systems; mission critical complex real-time systems; functional; behavioral; structural; resource allocation	15. NUMBER OF PAGES 46
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR